Self-Hosted Scripting in Guile

Fast Start with ELF and DWARF

Andy Wingo — Igalia, S.L.

@andywingo / wingolog.org

Self-hosted runtimes can be heavy

Wanted:

- Fast start
- Scripting source model
- Self-hosted runtime, compiler, interpreter

C: No source model (not scripting)

Lua: VM written in C (not self-hosted)

Guile: All three?

Challenge: Start Time

Fast start for:

- **Runtime**
- Interpreter/Compiler
- Debugger

LuaJIT does not have this problem; runtime in C, user code is first and only Lua

Tension with debuggability: metadata takes space, may take time

As program grows, more of it becomes “runtime”

ELF and DWARF help start time

Agenda:

- ELF and DWARF background
- How Guile uses ELF and DWARF
- Evaluation: Guile 2.2 (with ELF) vs 2.0 (without)

ELF

UNIX object file format

- Intermediate build products (`.o` files)
- Shared libraries for dynamic linking (`.so` files)
- Executables (standalone, or dynamically linked)

Two perspectives on ELF: loader vs inspection

Loading ELF

"What's the least work needed to load this `.so`?"

In Linux, system loader is `ld.so` by default

- Read fixed-size header, check it's ELF & right arch
- Read array of *segments* of file to `mmap` into memory
- Perform relocations, if needed

Compiler and linker's job to limit run-time relocation work

System loader not hard-coded!

Working with ELF

"What's in this damn thing?"

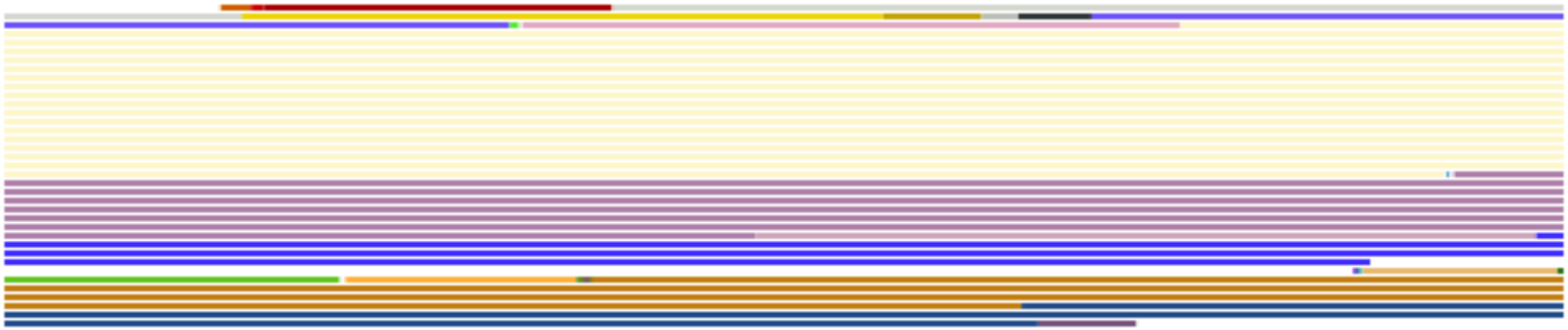
Array of named *section* descriptors at back of file

Sections may be in file but outside any segment: never mapped by loader

Some section names are well-known (`.data`, `.text`)

Open, extensible set of section names

ls



.interp	80
.note.ABI-tag	32
.hash	916
.dynsym	3120
.dynstr	1684
.gnu.version	260
.gnu.version_r	96
.rela.dyn	192
.rela.plt	2568
.init	23
.plt	1728
.text	70329
.fini	9
.rodata	26836
.eh_frame_hdr	2052
.eh_frame	11852
.init_array	8
.fini_array	8
.jcr	8
.dynamic	512
.got	16
.got.plt	880
.data	608
.comment	17
.bss	4552
.gnu_debuglink	16
.symtab	13416
.strtab	8237
.shstrtab	256

DWARF

UNIX debugging information format

Debugging information: ancillary metadata about program

Implementation: ELF sections with well-known names

What DWARF does

PC-to-source mapping

Inventory of functions and methods in text

Inventory of types used by text

Info about function arguments, locals, their scopes, etc

How to find locals in a function activation

How to find previous stack frame

DWARF design point

Ancillary: can be stripped from object file without changing semantics

- Links never go from text to debuginfo

Space-optimized

Speed of loading is important too (e.g. when debugging big C++ programs with GDB), but not primary

ELF and DWARF in Guile

Lazy caching compiler (think `.pyc`)

Guile compiler/linker emits ELF and DWARF

Guile loader loads Guile's ELF

Guile debugger reads DWARF

No dep on system linker/loader/debugger

Additional custom ELF sections for speed-sensitive side tables (e.g. stack map)

Loading in Guile

Map whole file as read-only

Read table of segments, making some private writable (`mprotect`)

Process directives in `PT_DYNAMIC` segment

- Check Guile VM version
- Find relocation thunk
- Add GC roots
- Find stack maps

Run relocation thunk

Benefits of ELF to Guile

- Static allocation of constants, other data
- Constants not needing relocation stay shareable and read-only
- Strippable debug info
- No heap-allocated metadata

Indirect benefits of ELF

Removal of procedure objects; no need for heap object to point at debug info

Support for unboxed locals and precise local lifetimes (raw / unused / live / dead slot map)

Closure optimization (no need for distinguished parameter 0)

GC implications

Loading adds GC roots

Guile-specific section for stack maps for precise stack GC

ELF mappings themselves not yet collectable

Future plans

Aggregating separately-compiled modules together (linker hack)

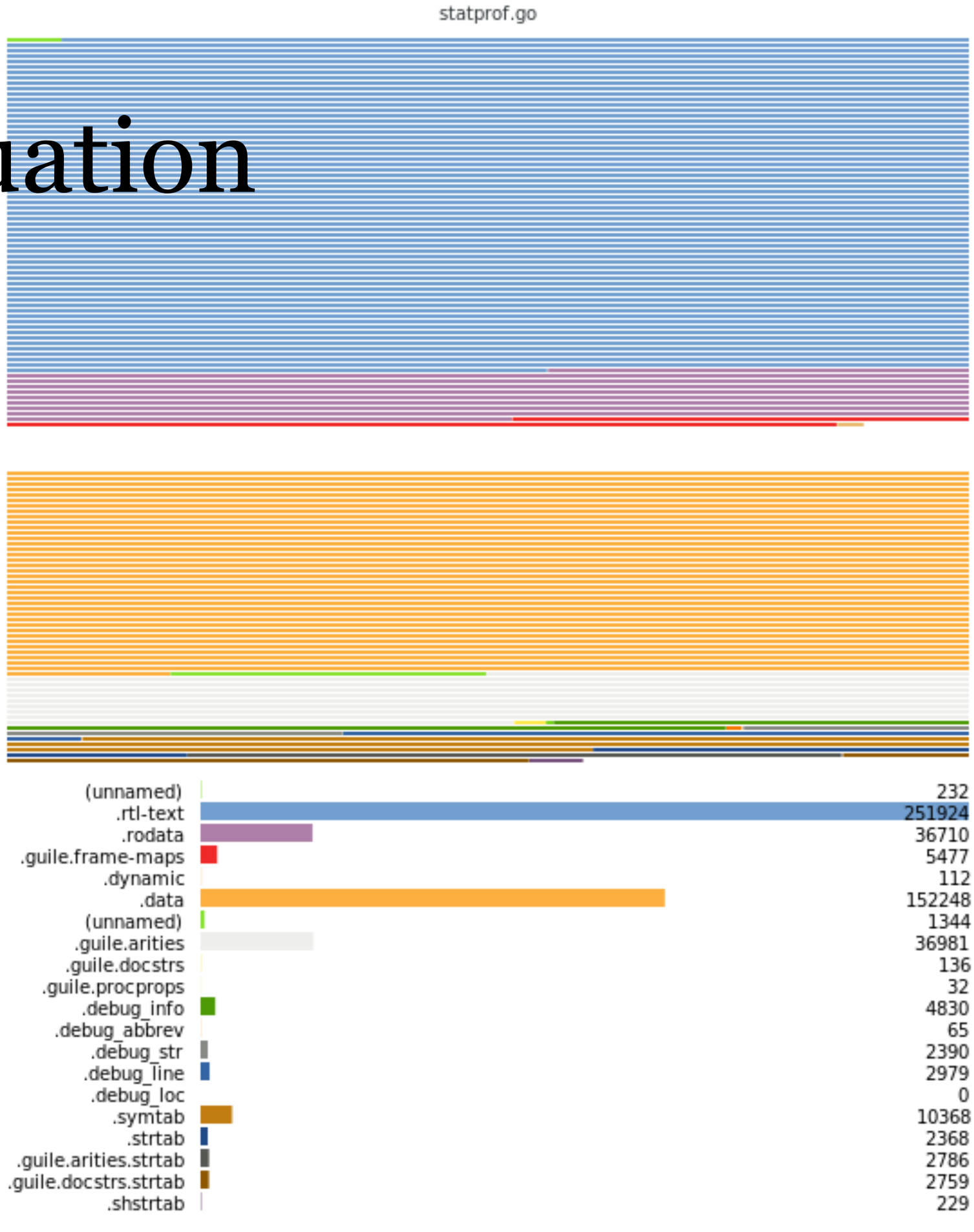
Linking static binary

Embed IR or source in object file?

AOT native code generation

All enabled by ELF's flexible sections and segments model

Evaluation


statprof.go
(unnamed) 232
.rtl-text 251924
.rodata 36710
.guile.frame-maps 5477
.dynamic 112
.data 152248
(unnamed) 1344
.guile.arities 36981
.guile.docstrs 136
.guile.procprops 32
.debug_info 4830
.debug_abbrev 65
.debug_str 2390
.debug_line 2979
.debug_loc 0
.symtab 10368
.strtab 2368
.guile.arities.strtab 2786
.guile.docstrs.strtab 2759
.shstrtab 229

Evaluation

`guile -c '(sleep 100)'`

Guile 2.0 (pre-ELF)

- 11 object files, 8.0e3 SLOC
- 12.5ms startup
- 3244 KB private dirty memory

Guile 2.2 (ELF and DWARF)

- 20 object files (+81%), 9.8e3 SLOC (+22%)
- 10.3ms startup (-18%)
- 2720 KB private dirty memory (-16%)

Summary

Dynamic VMs can start fast!

ELF and DWARF embody UNIX experience: how to minimize startup work

Steal the good ideas from ELF, but implement your own linker/loader/debugger

- http://gnu.org/s/guile/
- http://wingolog.org/
- @andywingo
- http://igalia.com/compilers/